Lab 6
Modified Instructions

EELE 467
Due date: 10/08/2024

Now that we've created our LED patterns hardware, we're going to start working towards being able to control our hardware from software. The first step is to create *registers* that software will be able to interact with. This lab has you create registers and put them on the HPS-to-FPGA Lightweight bus so software can access them via memory-mapped operations.

You'll create a new component called `led_patterns_avalon` that instantiates your `led_patterns` component and creates registers. You'll use Platform Designer—Quartus' system integration tool—to connect your `led_patterns_avalon` component to the HPS-to-FPGA Lightweight bus.

Outline

You'll perform the following high-level tasks:

1. Modify your existing Quartus project to include the HPS.
2. Create your led_patterns_avalon component:
   (a) Write the VHDL code.
   (b) Create an *IP (intellectual property) core* that can be used in Platform Designer.
3. Add your `led_patterns_avalon` IP core to your system using Platform Designer.
4. Verify that your design still functions as it did in Lab 4.

Lab 6 only verifies that your LED patterns hardware still functions correctly after creating your IP core. Lab 7 will test the registers.

Don't follow the instructions in the textbook for this lab. We are using a different Quartus project and entity.

Quartus Project Setup

In order to use the HPS, we need to

1. add signals to the entity that connect the HPS to on-board components such as DRAM, Ethernet, etc.
2. create pin assignments for the top-level HPS signals
3. add and configure the HPS in Platform Designer.
4. instantiate the Platform Designer system (which contains the HDL code that interfaces with and supports the HPS—remember, the HPS itself is "hardened" in silicon in the SoC FPGA IC.).

Rather than doing all the configuration and setup ourselves, we're leveraging a DE10-Nano reference design provided by Terasic (the `DE10_NANO_SoC_GHDR` project, to be specific). I translated their design from verilog to VHDL and fixed the reference design's incomplete pin assignments for you.

Entity and Pin Assignments

1. Update your entity using the `de10nano_entity.vhd` file available on GitHub.
2. Update your pin assignments by importing the `de10nano_pin_assignments.qsf` file available on GitHub.

Platform Designer

Although the HPS is "hardened" in silicon, the Quartus project still needs to configure the HPS and supporting peripherals; this is done through Platform Designer. We'll add a minimal Platform Designer *qsys* file to our project that includes a basic HPS configuration (Platform Designer used to be called QSys).

1. Download the `soc_system.qsys` file from GitHub and put in your `quartus` folder.
2. Open the `soc_system.qsys` file in Platform Designer.
3. Click "Generate HDL..." to generate HDL code for the system.
4. Add the generated `soc_system.qip` file to your Quartus project. The file is located at `soc_system/synthesis/soc_system.qip`. This file contains TCL (pronounced as "tickle") code to add and configure all the generated files to your project.

Instantiate the Platform Designer System

The `soc_system` component we generated from Platform Designer needs to be instantiated:

- copy the soc_system declaration into your top-level file.
- copy the soc_system instantiation into your top-level file; you'll need to map your reset signal to `reset_reset_n` in the port map.

Make sure your Quartus project compiles before moving on.

Creating the LED patterns Avalon Bus Wrapper

To make our LED patterns component accessible to software, we need to put it on a memory-mapped bus. The HPS-to-FPGA Lightweight bridge is the bus we'll use.

To put our component on the bus, we need to write VHDL code that defines the bus interface and how our component responds to bus transactions. In Intel FPGAs, there are two primary bus protocols: Avalon and AXI. AXI is the industry standard protocol for most embedded systems; indeed, the HPS in our FPGA uses the AXI protocol extensively. However, the Avalon protocol is generally simpler, so we'll use that instead.

Creating the VHDL Code

Create a `led_patterns_avalon.vhd` file in `hdl/led-patterns/` with the following entity:

```vhdl
entity led_patterns_avalon is
port (
    clk : in    std_ulogic;
    rst : in    std_ulogic;

    -- avalon memory-mapped slave interface
    avs_read      : in    std_logic;
    avs_write     : in    std_logic;
    avs_address   : in    std_logic_vector(1 downto 0);
    avs_readdata  : out   std_logic_vector(31 downto 0);
    avs_writedata : in    std_logic_vector(31 downto 0);

    -- external I/O; export to top-level
    push_button : in    std_ulogic;
    switches    : in    std_ulogic_vector(3 downto 0);
    led         : out   std_ulogic_vector(7 downto 0)
);
end entity led_patterns_avalon;
```

Listing 1: `led_patterns_avalon` entity. The Avalon memory-mapped slave signals all start with avs so Platform Designer can automatically interpret the signals as Avalon Slave signals.

Your `led_patterns_avalon` component will

- instantiate your `led_patterns` component.
- define registers and bus transactions.

Create the Registers

Read section 6.2.1 of the textbook to learn how to create registers. You'll create registers for the following signals:

- `hps_led_control`
- `base_period`
- `led_reg`

Creating the IP Core

Once the VHDL code for `led_patterns_avalon` has been written, you'll create an IP core for it using Platform Designer.

The steps for creating a Custom Platform Designer Component are as follows:

1. Open **Platform Designer** in Quartus and click the **New...** button in the IP Catalog panel (or select File → New Component). The Component Editor window will pop up.
2. In the **Component Type tab**, fill in the component information.
3. Click on the **Files tab**.

(a) Under the **Synthesis Files** section (not the VHDL or Verilog Simulation Files sections), click **Add File...**. Add your `led_patterns_avalon` file and all files it relies on. An IP core should be a single unit that contains all the files it needs.

(b) Set your `led_patterns_avalon.vhd` file as the top-level file by clicking in the "Attributes" field.

(c) Click the **Analyze Synthesis Files** button. You should see the green message **Analyzing Synthesis Files: completed successfully**. If you don't get this message, it usually means you have a VHDL syntax error in your VHDL code. You will need to correct this before proceeding.

(d) In the messages window, you will see some error messages that we will fix next, so ignore them for now.

4. Click on the **Signals & Interfaces tab**.

   (a) If Platform Designer didn't create clock or reset interfaces, create them as follows:

      i. Click *«add interface»* and add a Clock Input and/or Reset Input.

      ii. Under the Clock/Reset Input, click *«add signal»* to add your clock or reset signal.

      iii. You need to rename your clock and reset signal names to match what you put in your `led_patterns_avalon` entity.

   (b) Set the associated clock and associated reset for the `avalon_slave_0` interface. Doing this should remove the error messages saying "Interface must have an associated clock/reset".

   (c) It's possible that Platform Designer either didn't create or incorrectly created interfaces for our push button, switches, and led ports. These signals need to be exported to the top-level. We can do that with a *conduit* interface. Assuming Platform Designer didn't create an interface for your exported signals:

      i. Add a conduit interface and name it `export` (the name is arbitrary).

      ii. Add your `push_button`, `switches`, and `led` signals to the conduit interface. If the signal names don't show up, you can add them manually by clicking the * that pops up in the "add signal" dialog.

      iii. For each signal, make the "Signal Type" the same as the signal name. Set the width and in/out direction appropriately.

   (d) If there are still errors or warnings, fix them before moving on.

5. Click on the **Block Symbol tab**. You should see all the signals in your entity that have now been interpreted correctly.

6. Click the **Finish...** button. It will ask you if you want to save the .tcl script `led_patterns_avalon_hw.tcl` to your project directory. Click **Yes,Save**. The .tcl file is what allows the new custom component to show up in the IP Catalog panel when Platform Designer opens.

Instantiating your IP Core using Platform Designer

In Platform Designer and in the IP Catalog panel, under Project, you should now see your component.

1. Add the component to the Platform Designer system.

(a) Click on your component in the IP Catalog panel.

(b) Click on the "+ Add..." button

(c) Click Finish

2. Scroll down so you see the component and the bus/signal connection options.

   (a) Connect **clock** to **clk**. Highlight *clk* by clicking on *clk* in the `fpga_clk` component. In gray you will see that this can be connected to clock. Click on the small circle to make this connection, which will turn the circle black. Since we are adding the memory-mapped interface to the lightweight HPS bus, the clock really needs to be connected to the same clock that is feeding the `h2f_lw_axi_clock` clock input signal in the hps component; using the same clock for all components on the bus let's us avoid clock-domain crossings, which add extra complications and hardware to the system.

   (b) Connect **reset** to **clk_reset**

   (c) Connect the memory-mapped interface to the **h2f_lw_axi_master** port that is on the hps component. This connects the components registers to the lightweight HPS bus.

   (d) On the line that says **export** in the *Name* column and also **Conduit** in the *Description* column for your component, double click where it says **Double-click to export**. Rename this export signal name from *hps_led_patterns_0_export* to **led_patterns**.

3. Click the Platform Designer button **Generate HDL...** that is found at the lower right corner of Platform Designer.

4. From the Platform Designer menu bar, select Generate → Show Instantiation Template...

   (a) Select the HDL language to be **VDHL**.

   (b) Click the **Copy** button and paste into a text editor. This gives you the component declaration and instantiation template that needs to be placed in the top level of your Quartus project. **Note: Don't actually paste this into the top level.** We are only using it to see what changed. You don't want to reenter all the instantiation connections for the DRAM, etc.

   (c) Notice that there are three new signals in the soc_system component declaration. Add these signals into the soc_system component declaration in your top-level file.

   (d) These signals need to be connected at the top level to their associated ports in your top-level entity.

5. Click the Platform Designer button **Finish** that is found at the lower right corner of Platform Designer.

6. A pop-up window will remind you that the generated .qip file needs to be added to the project.

7. Add the .qip file to your Quartus project. The file is located at `soc_system/synthesis/soc_system.qip`.

When you generate HDL from Platform Designer, Platform Designer copies all your VHDL files into `soc_system/synthesis/submodules`. The .qip file uses your files at `soc_system/synthesis/submodules` when compiling the project. If you need to make changes to your files, you should make the changes in your original source directory (`hdl/`) and then regenerate HDL from Platform Designer. You can directly edit the "copied" files to save time, but then you have to remember to copy your changes back to your original source files; I don't recommend doing this, as you will eventually forget to copy the changes, regenerate HDL from Platform Designer, and accidentally overwrite your changes!

Once you've updated your `soc_system` instantiation to include your `led_patterns` export signals, you need to remove your instantiation of your `led_patterns` component from your top-level file; if you don't do this, you'll have multiple LED patterns components trying to drive the LEDs, which won't work.

Compile your Quartus project, program your board, and make sure your LED patterns still work as they did in Lab 4.

Cleaning up the IP Core

When you saved your `_hw.tcl` file your IP core, Quartus saved the file in the Quartus project directory. It would be cleaner to have the IP core's tcl file in the same directory as the rest of the core's files (i.e., in `hdl/led-patterns`). We can easily move the file to that location, but we need to adjust some file paths in the tcl file as well as tell Quartus where to search for the IP core.

1. Move your `_hw.tcl` file to `hdl/led-patterns`.
2. Open the tcl file in a text editor and change all the `add_fileset_file` commands to reference your HDL files relative to your `hdl/led-patterns` folder.
3. Create an `soc_system.ipx` file in your Quartus project directory with the following contents. The ipx file tells Platform Designer where to search for IP cores.

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <path path="../hdl/**/*"/>
</library>
```

Once you've done the above, reopen Platform Designer and verify that your IP core still shows up. Regenerate your HDL and recompile your Quartus project to verify that everything is still working. If everything is working, you're done! 🎉

Deliverables

Lab Report

You'll write a lab report in `docs/lab6.md`.

Your lab report must contain the following sections, which are described in more detail below.

1. System Architecture
2. Register Map
3. Platform Designer

System Architecture

Brifely describe your system architecture. Include an updated block diagram that shows your Avalon bus wrapper component and the registers your created.

Register Map

Include a description of the registers that you created and how they are read from and written to. Additionally, create bitfield diagrams showing what each bit of your 32-bit registers correspond to. You may consider using the bit-field syntax for wavedrom to create your bitfield diagrams.

Create a table with your address map; that is, show what address each register is at.

Platform Designer

Answer the following questions:

- How did you connect these registers to the ARM CPUs in the HPS?
- What is the base address of your component in your Platform Designer system?

GitHub Submission

Follow the workflow given in your repository (`docs/workflow.md`), per usual. Make sure all your files are committed.